Programming Languages and Paradigms

COMP 302, Fall 2016

Assignment 3

Due date: Monday, November 14, 2016
6pm

In this assignment you will develop the code to evaluate the WML language parsed in the previous assignment. You will therefore need an AST generator. You may use your code from assignment 2, or base this assignment on the code provided as a solution to assignment 2, available in mycourses.

Again, a(n optional) framework you can use to test your evaluator code is provided through the files `wml.html` and `wml.js`, or you may adapt the evaluator code written by some of your TAs and linked in *MyCourses*.

Your code must run without error or modification in current versions of Firefox or Chrome: restrict yourself to basic JavaScript idioms defined in the ECMA standard 5.1. Be sure to respect precise naming, input, and output requirements.

All code should be well-commented, in a professional style, with appropriate variables names, indenting (uses spaces and avoid tabs), etc. **The onus is on you to ensure your code is clear and readable. Marks will be very generously deducted for bad style or lack of clarity.**

1. Develop code to represent environments. Environments are objects, with a (probabilistically) unique name as a floating point number given by `Math.random()`, a set of bindings, and a parent environment (object, not name). **3**

   ```
   {  name:  ,
      bindings:  { ...  },
      parent:  }
   ```

   You should have have 2 functions associated with this structure,

   - `createEnv(parent)`, which creates a new environment.
   - `lookup(name,env)`, which returns the first binding value for binding key `name`. This function searches through the given environment and then parents, returning `null` if no binding is found.

2. Develop functionality to evaluate the AST generated from your solution to assignment 2. You should define a set of functions for processing the different AST nodes, each of which received an AST node and an environment, and returns a string. **15**

   At the top-level you should define function `evalWML(ast,env)`. This function should be able to process a (list of) `outer` AST nodes, evaluating text or delegating to other, appropriate `eval*` functions as appropriate.

   so it only evaluates outer --> which contains template invoc and intemplate def

   Your code should add appropriate bindings at template definitions. A definition binding is a map between a template name, and a structure which records the parameters as an array, the body AST node, and the defining environment.

   ```
   {  params:  ,
      body:  , --> body link to a AST node or...?
      env:  }
   ```

   Your code from this question should be capable of correctly executing non-trivial WML code, including invocations, nested definitions, and recursive definitions. You should aim to implement static scoping.

3. To be able to easily and efficiently do non-trivial computation, you will need some special functions. **5**

Modify your evaluation code to recognize `#if`, `#ifeq`, and `#expr` when invoked and perform the appropriate behaviour.

4. Add the ability to return and pass around a template declaration as a first-class value. As described in class, this is done by recognizing the single-backquote ` character as the first character in a template name (but is excluded from the actual name of the template). **10**

   In this your code should allow for anonymous templates to be returned, expressed as templates with no name (other than the `.). Note that returned templates with names are still recorded as bindings in the current environment, but anonymous templates are not.

   To return a template, convert the template binding value to a string. When a returned template is subsequently encountered in an invocation, it must be converted back to a binding value. Use the `stringify` and `unstringify` functions provided (these are for your convenience, you may develop your own).

   Note that there is some uncertainty in an invoke as to whether the template being invoked is specified by name or by a binding.

5. Show how to compute prefix-sum with Church numerals in WML. Define suitable helper templates, such as for `succ`, `isZero`, `plus` as well as `makeN`, which receives an argument `n` and returns the Church numeral for `n`. Recall that prefix-sum computes $\sum_{i=1}^{n} i$ for an input n. **7**

What to hand in

Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

For each question n, include a file `qn.js` with the source code of your answer. You can assume your code from question 1 will be included in evaluating questions 2–4. Do not include the provided files (`wml.html` and `wml.js`), or `stringify.js`, unless you altered it. If you did not use the provided solution to assignment 2, include am `assig2.js` file with your code.

This assignment is worth 10% of your final grade. **40**